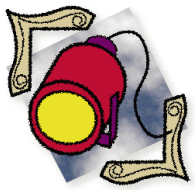


Delphi 2 / Object Pascal

By Michael Maloof

An Eye on GDI

A Delphi 2 Object for Monitoring the Wily GDI Beast

It's no secret: most programmers know that Windows 95 is not a true 32-bit operating system. Everyone seems to appreciate that some compromises were necessary to ensure compatibility with existing 16-bit applications. What may surprise you is that one of those 16-bit compromises can seriously affect the stability of your Delphi 2 applications.

Tracking your application's GDI (Graphics Display Interface) usage, and the available GDI resources, are common tasks for Windows 3.1 and Delphi 16-bit development. It's all too easy to build spectacular forms with layers of panels, tabs, and grids. Running out of memory is just as easy. Often the GDI is exhausted, with symptoms ranging from partially displayed screens to a completely frozen machine.

Woebegone Days

With Windows 95, Delphi 2, and the multi-gigabyte, flat-address-space world of 32-bit programming, those GDI woes are gone forever, right? Not quite. Windows 95 has done much to improve the limitations of Windows 3.1 programming, but the GDI remains true to its 16-bit heritage.

The bad news is the GDI is still a 16-bit resource and remains limited to 64K. However, the good news is that Windows 95 uses less GDI memory than Windows 3.1 by shifting items such as TrueType font support to other areas of the system.

Is a 64K GDI region still a serious issue? It can be if left unmonitored. The point is that you need to know your program's impact on the GDI, and you should keep an eye on the system-wide GDI usage. This is easily done using one of the commercial utilities on the market. For example, Norton Utilities for Windows 95 contains a sophisticated resource monitor that can be set to warn you of impending doom. There are even shareware and freeware monitors available (one is included in the Accessories folder with Windows 95).

Delphi: An Ideal Custom Solution

While these external monitors can do the job, the ideal solution is to create a Delphi-based GDI resource monitor that can be linked to your application. The Delphi approach allows you to track your GDI usage with much finer granularity. During development, you can observe the GDI impact of specific control creation events by simply calling your GDI monitor before and after the event.

A Delphi-based GDI monitor could even be used to protect your application from other GDI-eating applications. Simply check the available GDI memory at critical points in your application, and warn the user to close a few applications or windows before continuing.

Building a GDI resource monitor in Delphi should be simple. In fact, a few 16-bit versions of Delphi-based resource monitors already exist. The problem is that the one function used to measure the GDI, namely *GetFreeSystemResources*, is no longer available. (At least, it's no longer part of the published Win32 API.)

From the Microsoft point of view, *GetFreeSystemResources* is no longer support-

ed. Its replacement, *GlobalMemoryStatus*, is a useful function that returns a wealth of information, but absolutely nothing about GDI usage. This is unfortunate considering the importance of this resource.

Get Me Some Free Resources

Of course, the *GetFreeSystemResources* function isn't really gone. It still exists as part of the Windows 95 support for 16-bit applications. You'll find the function in the file USER.EXE. The question is, how do you call this 16-bit function from your 32-bit Delphi application?

There are three ways this could be accomplished. The Microsoft approved method for calling a 16-bit routine from a 32-bit Windows 95 application is to use a *flat thunk*. The process is actually uglier than its name, and involves the use of a *thunk* compiler and separate 16- and 32-bit DLLs to perform this 16-to-32-bit magic. In spite of being the approved method, this only works under Windows 95, meaning that your code must compensate for running under Windows NT.

The undocumented method — which happens to be the one Microsoft uses — involves the use of the *LoadLibrary16*, *FreeLibrary16*, and *GetProcAddress16* functions found in the KERNEL32.EXE. Matt Pietrik discovered these undocumented functions, and described their use in his "PC Tech" column in the September 26, 1995 issue of *PC Magazine*. Considering Microsoft's reliance on these functions, they're probably safe to use, but like the flat thunk, they'll work only on Windows 95.

The easy method — which happens to be the one I use — takes advantage of the fact that Delphi 2 includes the original Delphi 16-bit program. You can use the 16-bit version to build a very small, non-visual program that does nothing but call *GetFreeSystemResources* and return the GDI utilization. This 16-bit application is called from inside your 32-bit Delphi application, and the return value can be processed any way you see fit.

This technique works equally well under Windows 95 and Windows NT. (It should be noted that Windows NT, as a true 32-bit operating system, does not have a GDI usage problem. The fact that this technique works under NT simply means that your code can safely call the GDI monitor without worrying about the current platform.)

Credit for the easy method must be given to Lou Grinzo, who presents the technique in his book *Zen of Windows 95 Programming* [The Coriolis Group, Inc., 1996].

Getting Started

The first step is to create the 16-bit application that will call *GetFreeSystemResources* and return the current GDI utilization. Let's step through the process:

- Launch the 16-bit version of Delphi and start a new project. Then use the Project Manager to remove the *Unit1* file from the default *Project1*. The resulting project source code is shown here:

```
program Project1;

uses
  Forms;

{$R *.RES}

begin
  Application.Run;
end.
```

- This leaves a project with only one unit: *Project1*. Edit *Project1* and remove the *Forms* unit from the **uses** clause. By removing the *Forms* unit, you'll find that Delphi can produce a very small executable.
- Add the *WinTypes* and *WinProcs* units to the **uses** clause. These units give us the access we need to the Windows 16-bit API and the associated constants.
- Finally, remove the *Application.Run* statement, and replace it with a *Halt* statement that calls the *GetFreeSystemResources* function and passes our request for the GDI utilization. The result is shown in Figure 1. The *Halt* statement tells an application to terminate, but it can also pass back an exit code to the application that launched it. In this case, our exit code is the amount of free GDI resources.

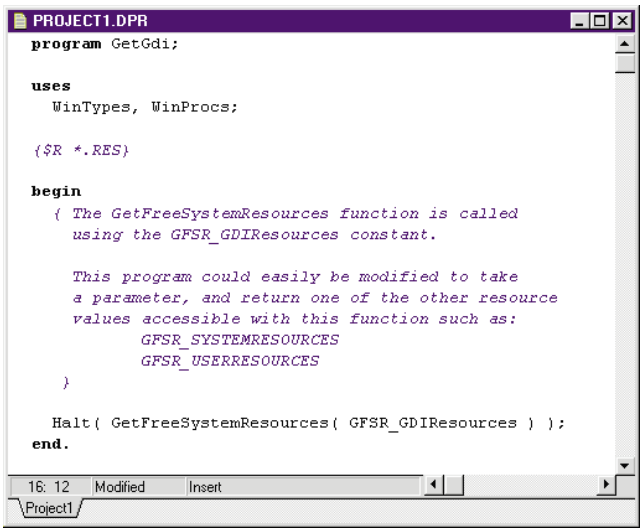
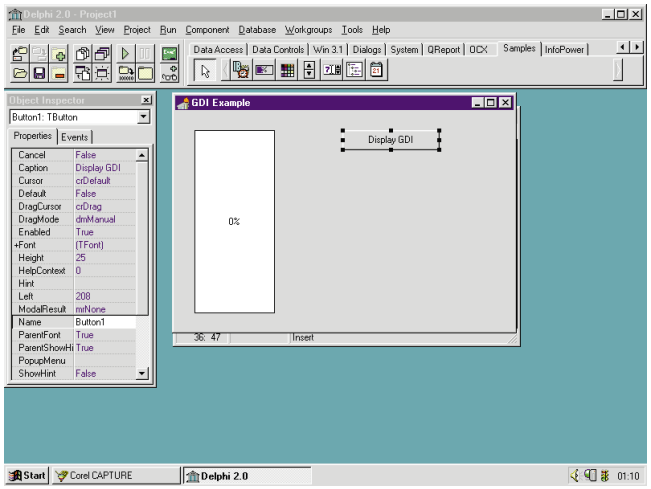


Figure 1: This simple Delphi 1 project makes a call to the Windows 3.x API function, *GetFreeSystemResources*, and returns the result to the program that calls it.

A Simple Example

The next step is to integrate this executable into a 32-bit Delphi application. To illustrate the basic principle, we'll create a simple form that contains a Gauge and a Button. When the Button is pressed, the Gauge progress value will be set to the available GDI.

Using a default project and form, place a Gauge component on the form. In our example, the Gauge's *Kind* property was set to *gkVerticalBar*. Place a Button on the form, and change the Button's *Caption* property to **Display GDI** (see Figure 2). Once the Gauge and Button have been placed on the form, double-click on the Button to create the framework for the *ButtonClick* procedure and enter the code shown in Figure 3.




```
procedure TForm1.Button1Click(Sender: TObject);
var
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
  GdiValue: DWORD;
  cp: Boolean;
begin
  FillChar(StartupInfo, SizeOf(TStartupInfo), 0);

  // Get GDI resource utilization by calling
  // our GETGDI.EXE 16-bit Delphi application.
  cp := CreateProcess(nil,'GETGDI.EXE', nil, nil, False,
                      NORMAL_PRIORITY_CLASS, nil, nil,
                      StartupInfo, ProcessInfo);
  if cp then
    begin
      WaitForSingleObject(ProcessInfo.hProcess, 1000);
      GetExitCodeProcess(ProcessInfo.hProcess, GdiValue);
      CloseHandle(ProcessInfo.hProcess);
      Gauge1.Progress := GdiValue;
    end
  else
    Gauge1.Progress := 0;

end;
```

Figure 2 (Top): Building a simple GDI monitor with Delphi 2 that calls the Delphi 1 application shown in Figure 1.
Figure 3 (Bottom): The sample application's *ButtonClick* procedure.

You'll notice we're using a call to *CreateProcess* to launch our 16-bit application, GETGDI.EXE. Windows 95 will go through the following specific search sequence to find this executable:

- The directory from which the main application was loaded.
- The current directory.
- The Windows System directory.
- The Windows directory.
- The directories that are listed in the PATH environment variable.

When you run this simple example and press the Button, the current GDI-free percentage is returned from our GETGDI executable. The Gauge reflects this percentage. Assuming that Delphi is still running, you may notice that your GDI-free percentage is already in the 80s. Considering that you were probably in the high 90s when Windows 95 started, you've already lost a fair amount of memory.

A GDI Monitoring Object

Where did the GDI go? While prototyping a new Delphi 2 application, I was rudely awakened by Norton's System Doctor warning me that my GDI resources were dangerously low. This led to the discovery of the 16-bit nature of the Windows 95 GDI, but that was only the beginning.

Knowing that the GDI is still a limited resource and knowing how to preserve that resource are two distinct issues. Which form consumed the most GDI? Which components or controls could be eliminated? Which forms should be created during program initialization, and which ones should be created only on demand? How many forms can be opened before the application approaches critical mass?

The search for these answers resulted in the creation of a *TTimeMem* object. Our lead Delphi programmer, Lynn Settle, constructed the *TTimeMem* object to measure the elapsed time and the memory consumed between two events. [*TTimeMem* is available for download from Library 7 in the Delphi Forum on CompuServe. The file name is TIMEMEM.ZIP.] It was during the creation of the *TTimeMem* object that we discovered there was no apparent way to track the GDI.

Using the technique presented in this article, we now have a *TTimeMemGDI* object that adds GDI tracking to the original object's capabilities. This object can be integrated into your application in many ways.

To illustrate the use of the object, it's included in the main project source file shown in Figure 4. By incorporating *TTimeMemGDI* in this fashion, we can now allow Delphi to create the sample form, and measure the form's impact on the GDI, as well as other system resources (see Figure 5). The complete .PAS file for *TTimeMemGDI* is shown in Listing One, starting on page 21.

```
program Project2;

uses
  Forms,
  TimeMemG,
  Unit2 in 'Unit2.pas' {Form1};

{$R *.RES}

var
  TimeMem : TTimeMemGDI;
begin
  Application.Initialize;
  // Create an instance of the object.
  TimeMem := TTimeMemGDI.Create;
  // Capture starting values.
  TimeMem.Start;
  // Create the sample form.
  Application.CreateForm(TForm1, Form1);
  // Capture the ending values.
  TimeMem.Stop;
  // Show the impact of creating this form.
  TimeMem.DisplayInfo('Creating Sample Form');

  Application.Run;

end.
```

Figure 4: Using the *TTimeMemGDI* object.

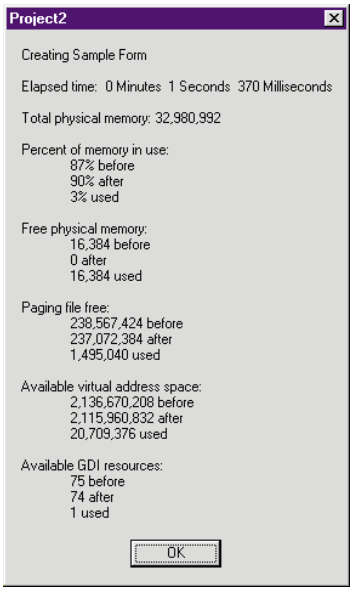


Figure 5: An example of the GDI Monitor at run time.

By creating an instance of the *TTimeMemGDI* object and calling the *Start* method, we have an accurate picture of the initial environment. The *Stop* method is called immediately following the *CreateForm* procedure, and the *DisplayInfo* method gives a complete picture of what has changed between the two events.

You can use the *TTimeMemGDI* object by calling sequences of *Start*, event of interest, *Stop*, and *DisplayInfo*. You can also measure the increasing impact of a series of events using a sequence such as, *Start*, event #1, *Stop*, *DisplayInfo*, event #2, *Stop*, *DisplayInfo*. By continually calling only the *Stop* and *DisplayInfo* methods, you'll see the cumulative effect of the events.

Conclusion

We concentrated our GDI resource reduction efforts on descendants of the *TWinControl*. We eliminated virtually all of our *TPanel* components using Eric Uber's *TGraphicPanel*, which descends from *TBevel,* but has the same visual properties as a *TPanel*. We also substantially reduced the number of *TDBNavigator* components. Furthermore, we carefully monitor the GDI situation as we create forms on demand. [Eric's *TGraphicPanel* is found in the file, GRAPHPNL.ZIP, located in Library 9 of the Delphi Forum on CompuServe.]

By using this technique we reduced the GDI usage in our application from an unbelievable 90 percent to about 30 percent, or roughly about twice as much as Delphi itself. That's not bad for a major vertical market application that relies heavily on Grid, Page, and Tab Controls. This reduction is especially significant because we'll soon be applying for Windows 95 Logo Certification.

One element of the certification process is a general stability test. Figure 6 shows the application being exercised while running Microsoft's Stress utility (available on the MSDN CDs). Among other things, the Stress utility randomly consumes between 40 and 60 percent of the GDI. An application that consumes more than 40 percent of the GDI may not survive in that environment. Will your application? [For more information about the Windows 3.x Stress utility, see Karl Thompson's article "Gimme Some Stress!" beginning on page 38.]

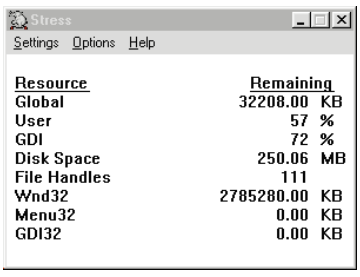


Figure 6: The Windows Stress utility at run time.

It's unfortunate that the GDI beast still haunts Windows 95, but with a little effort you can prevent your Delphi applications from becoming its next meal. Δ

The demonstration files referenced in this article are available on the Delphi Informant Works CD located in INFORM\96\JUL\DI9607MM.

Michael Maloof is CEO of Smart Shop Software, Inc., a Coeur d'Alene, ID-based publisher of software for the manufacturing industry. Following Jeff Duntemann's software publishing advice, Michael married his best friend, moved to a beautiful place, and brings his golden retriever to work every day. You can reach Michael through CompuServe at 76050,1607.

Begin Listing One: TTimeMemGDI

```
unit TimeMemG;

interface

uses SysUtils, Windows, Dialogs, Classes, Forms;

type
  TTimeMemGDI = class(TObject)
  private
    StartTime: TDateTime;
    UsedTime: TDateTime;
    // Bytes of physical memory.
    TotalPhys: Integer;
    // Percent of memory in use.
    StartMemoryLoad: Integer;
    EndMemoryLoad: Integer;
    // Bytes physical memory available.
    StartAvailPhys: Integer;
    EndAvailPhys: Integer;
    // Bytes available in paging file.
    StartAvailPageFile: Integer;
    EndAvailPageFile: Integer;
    // Bytes available in the user mode portion of
    // the virtual address space.
    StartAvailVirtual: Integer;
    EndAvailVirtual: Integer;
    // Percent GDI resources available.
    StartAvailGDI: Integer;
    EndAvailGDI: Integer;

  public
    // Capture start time, memory stats, and GDI free.
    procedure Start;
    // Capture stop time, memory stats, and GDI free.
    procedure Stop;
    // Display information.
    procedure DisplayInfo(InfoTitle: String);
  end;

implementation

procedure TTimeMemGDI.Start;
var
  SysMemoryStatus: TMemoryStatus;
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
  GdiValue: DWORD;
  cp: Boolean;
```

```
begin
  FillChar(SysMemoryStatus, SizeOf(TMemoryStatus), 0);
  //SysMemoryStatus.dwLength := SizeOf(TMemoryStatus);
  GlobalMemoryStatus(SysMemoryStatus);

  { Get starting memory info. }
  StartMemoryLoad    := SysMemoryStatus.dwMemoryLoad;
  StartAvailPhys     := SysMemoryStatus.dwAvailPhys;
  StartAvailPageFile := SysMemoryStatus.dwAvailPageFile;
  StartAvailVirtual  := SysMemoryStatus.dwAvailVirtual;

  FillChar(StartupInfo, SizeOf(TStartupInfo), 0);

  // Get starting GDI resources percentage.
  cp := CreateProcess(nil,'GETGDI.EXE', nil, nil, False,
                      NORMAL_PRIORITY_CLASS, nil, nil,
                      StartupInfo, ProcessInfo);
  if cp then
    begin
      WaitForSingleObject(ProcessInfo.hProcess, 1000);
      GetExitCodeProcess(ProcessInfo.hProcess, GdiValue);
      CloseHandle(ProcessInfo.hProcess);
      StartAvailGDI := GdiValue;
    end
  else
     StartAvailGDI := -1;

  // Before I go, brother could you spare the time?
  StartTime := NOW;
end;

procedure TTimeMemGDI.Stop;
var
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
  GdiValue: DWORD;
  cp: Boolean;
  SysMemoryStatus: TMemoryStatus;
begin
  // How long has it been?
  UsedTime := NOW - StartTime;

  SysMemoryStatus.dwLength := SizeOf(SysMemoryStatus);
  GlobalMemoryStatus(SysMemoryStatus);

  // Get ending memory info.
  TotalPhys        := SysMemoryStatus.dwTotalPhys;
  EndMemoryLoad    := SysMemoryStatus.dwMemoryLoad;
  EndAvailPhys     := SysMemoryStatus.dwAvailPhys;
  EndAvailPageFile := SysMemoryStatus.dwAvailPageFile;
  EndAvailVirtual  := SysMemoryStatus.dwAvailVirtual;

  FillChar(StartupInfo, SizeOf(TStartupInfo), 0);

  // Get ending GDI resources percentage.
  cp := CreateProcess(nil,'GETGDI.EXE', nil, nil, False,
                      NORMAL_PRIORITY_CLASS, nil, nil,
                      StartupInfo, ProcessInfo);
  if cp then
    begin
      WaitForSingleObject(ProcessInfo.hProcess, 1000);
      GetExitCodeProcess(ProcessInfo.hProcess, GdiValue);
      CloseHandle(ProcessInfo.hProcess);
      EndAvailGDI := GdiValue;
    end
  else
     EndAvailGDI := -1;
end;

{ The DisplayInfo procedure allows you to visually
  inspect the values captured by the Start and Stop
  methods. The InfoTitle parameter allows you to
  display a meaningful title, such as 'After Order Entry
  Form Creation.' This approach was adequate for our
  needs, but certainly leaves room for significant
  enhancements.}
```

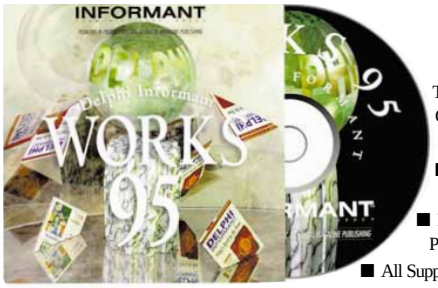
```
procedure TTimeMemGDI.DisplayInfo(InfoTitle: string);
const
  CRLF = #10+#13;
  TAB  = #9;
var
  Hour, Min, Sec, MSec: Word;
var
  TimeMemMessage,
  StrUsedMemoryLoad,
  StrUsedAvailPhys,
  StrUsedAvailPageFile,
  StrUsedAvailVirtual,
  StrUsedAvailGDI,
  StrStartMemoryLoad,
  StrStartAvailPhys,
  StrStartAvailPageFile,
  StrStartAvailVirtual,
  StrStartAvailGDI,
  StrEndMemoryLoad,
  StrTotalPhys,
  StrEndAvailPhys,
  StrEndAvailPageFile,
  StrEndAvailVirtual,
  StrEndAvailGDI:          string;
begin
  DecodeTime(UsedTime, Hour, Min, Sec, MSec);

  StrUsedMemoryLoad :=
    IntToStr(EndMemoryLoad - StartMemoryLoad);
  StrUsedAvailPhys :=
    FloatToStrF((StartAvailPhys - EndAvailPhys),
                ffNumber, 10, 0) ;
  StrUsedAvailPageFile :=
    FloatToStrF((StartAvailPageFile - EndAvailPageFile),
                ffNumber, 10, 0) ;
  StrUsedAvailVirtual :=
    FloatToStrF((StartAvailVirtual - EndAvailVirtual),
                ffNumber, 10, 0) ;
  if (StartAvailGDI = -1) or (EndAvailGDI = -1) then
    StrUsedAvailGDI := 'Unable to determine'
  else
    StrUsedAvailGDI :=
      FloatToStrF((StartAvailGDI - EndAvailGDI),
                  ffNumber, 10, 0) ;
  StrTotalPhys :=
    FloatToStrF(TotalPhys, ffNumber, 13, 0);
  StrStartMemoryLoad :=
    FloatToStrF(StartMemoryLoad, ffNumber, 13, 0);
  StrStartAvailPhys :=
    FloatToStrF(StartAvailPhys, ffNumber, 13, 0);
  StrStartAvailPageFile :=
    FloatToStrF(StartAvailPageFile, ffNumber, 13, 0);
  StrStartAvailVirtual :=
    FloatToStrF(StartAvailVirtual, ffNumber, 13, 0);
  if StartAvailGDI = -1 then
    StrStartAvailGDI := 'Error accessing GetGDI.exe'
  else
    StrStartAvailGDI :=
      FloatToStrF(StartAvailGDI, ffNumber, 13, 0);
  StrEndMemoryLoad :=
    FloatToStrF(EndMemoryLoad, ffNumber, 13, 0);
  StrEndAvailPhys :=
    FloatToStrF(EndAvailPhys, ffNumber, 13, 0);
  StrEndAvailPageFile :=
    FloatToStrF(EndAvailPageFile, ffNumber, 13, 0);
  StrEndAvailVirtual :=
    FloatToStrF(EndAvailVirtual, ffNumber, 13, 0);
  if EndAvailGDI = -1 then
    StrEndAvailGDI := 'Error accessing GetGDI.exe'
  else
    StrEndAvailGDI :=
      FloatToStrF(EndAvailGDI, ffNumber, 13, 0);

  TimeMemMessage :=
    'Elapsed time:  ' + IntToStr(Min) + ' Minutes  '  +
    IntToStr(Sec) + ' Seconds  ' + IntToStr(MSec) +
```

```
    ' Milliseconds' + CRLF+CRLF +

    'Total physical memory: ' + StrTotalPhys +CRLF+CRLF+
    'Percent of memory in use:' +CRLF+
      TAB + StrStartMemoryLoad + '% before' +CRLF+
      TAB + StrEndMemoryLoad   + '% after' +CRLF+
      TAB + StrUsedMemoryLoad  + '% used' +CRLF+CRLF+

    'Free physical memory:' +CRLF+
      TAB + StrStartAvailPhys  + ' before' +CRLF+
      TAB + StrEndAvailPhys    + ' after' +CRLF+
      TAB + StrUsedAvailPhys   + ' used' +CRLF+CRLF+

    'Paging file free:' +CRLF+
      TAB + StrStartAvailPageFile  + ' before' +CRLF+
      TAB + StrEndAvailPageFile    + ' after' +CRLF+
      TAB + StrUsedAvailPageFile   + ' used' +CRLF+CRLF+

    'Available virtual address space:' +CRLF+
      TAB + StrStartAvailVirtual + ' before' +CRLF+
      TAB + StrEndAvailVirtual   + ' after' +CRLF+
      TAB + StrUsedAvailVirtual  + ' used' +CRLF+CRLF+

    'Available GDI resources:' +CRLF+
      TAB + StrStartAvailGDI + ' before' +CRLF+
      TAB + StrEndAvailGDI   + ' after' +CRLF+
      TAB + StrUsedAvailGDI  + ' used';

  ShowMessage(InfoTitle + CRLF+CRLF + TimeMemMessage);
end;

end.
```

End Listing One

The Must Have Reference Source For The Serious Delphi™ Developer

The Entire Text of Over 60 Technical Articles Appearing in Delphi™ Informant® in 1995



The Delphi™ Informant® Works 1995 CD-ROM Includes:

- Over 60 Technical Articles
- Text and Keyword Search Capability
- Improved Speed and Performance
- All Supporting Code and Sample Files
- 16-Page Color Booklet
- Electronic Version of *Delphi Power Tools* Catalog
- Electronic Version of *Borland C++ Power Tools* Catalog
- Third-Party Add-In Product Demos
- CompuServe Starter Kit with $15 Usage Credit.

A $105 Value Available Now for only

$39.95

California residents add 7¼% Sales Tax, plus $5 shipping & handling for US orders. (International orders add $15 shipping & handling)

Call Now Toll Free 1-800-88-INFORM

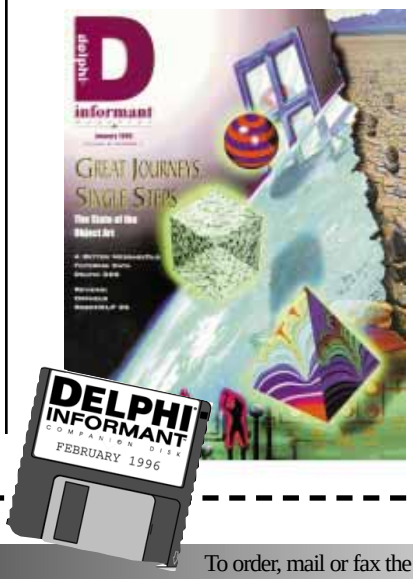
1-800-884-6367 Ask for offer # PI95 To order by mail, send check or Money Order to:
Informant Communications Group, Inc.
ATTN: Works CD offer # PI95
10519 E. Stockton Blvd, Suite 142 Elk Grove, CA 95624-9704
or Fax your order to
916-686-8497

Get Informed!

Subscribe to Delphi Informant, The Complete Monthly Guide to Delphi Development and stay ahead of the rapid application development curve.

Order Now and Get One Issue FREE!

For a limited time you can receive the first issue FREE plus 12 additional issues for only $49.95 That's nearly 25% off the yearly cover price!



Each big issue of *Delphi Informant* is packed with Delphi tips, techniques , news, and more!

- Client/Server Application Development
- Delphi Programming
- Using the Borland Database Engine
- Object -Oriented Programming
- Creating Reusable Components
- Product Reviews
- News from the Delphi Community
- Delphi User Group Information

Payment Method...

- ❑ Check (payable to Informant Communications Group)
- ❑ Purchase Order-- Provide Number _____
- ❑ Visa
- ❑ Mastercard
- ❑ American Express

_____
Card Number

_____
Expiration Date

_____
Signature

International rates

Magazine-only
$54.95/year to Canada
$74.95/year to Mexico
$79.95/year to all other countries

Magazine AND Companion Disk Subscriptions
$124.95/year to Canada
$154.95/year to Mexico
$179.95 to all other countries

California Residents add 7¼% sales tax on disk subscription

☐ YES

To order, mail or fax the form below or call (916) 686-6610 Fax: (916) 686-8497

I want to sharpen my Delphi Programming skills. I've checked the subscription plan I'm interested in below

☐ **Magazine-Only Subscription Plan...**
13 Issues Including One Bonus Issue at $49.95.

☐ **Magazine AND Companion Disk Subscription Plan...**
13 Issues and Disks Including One Bonus Issue and Disk at $119.95
The Delphi Informant Companion Disk contains source code, support files, examples, utilities, samples, and more!

☐ **Delphi Informant Works 1995 CD-ROM $39.95**
US residents add $5 shipping and handling. International customers add $15 shipping and handling.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

Country _____ Phone _____

FAX _____ e-Mail _____

PI95